
sphinxcontrib-autoprogram Documentation

Release 0.1.4

Hong Minhee

Feb 27, 2018

Contents

1	.. autoprogram:: directive	3
1.1	cli.py	3
1.2	subcmds.py	4
2	Additional Options for .. autoprogram::	7
3	Author and license	9
4	Changelog	11
4.1	Version 0.1.4	11
4.2	Version 0.1.3	11
4.3	Version 0.1.2	12
4.4	Version 0.1.1	12
4.5	Version 0.1.0	12
	Python Module Index	13

This contrib extension, `sphinxcontrib.autoprogram`, provides an automated way to document CLI programs. It scans `argparse.ArgumentParser` object, and then expands it into a set of `.. program::` and `.. option::` directives.

In order to use it, add `sphinxcontrib.autoprogram` into `extensions` list of your Sphinx configuration file (`conf.py`):

```
extensions = ['sphinxcontrib.autoprogram']
```

See also:

Module `argparse` This extension assumes a program to document is made using `argparse` module which is a part of the Python standard library.

`.. autoprogram:: directive`

Its only and simple way to use is `.. autoprogram:: directive`. It's similar to `sphinx.ext.autodoc` extension's `.. automodule::` and other directives.

For example, given the following Python CLI program (say it's `cli.py`):

```
import argparse

parser = argparse.ArgumentParser(description='Process some integers.')
parser.add_argument('integers', metavar='N', type=int, nargs='+',
                    help='An integer for the accumulator.')
parser.add_argument('-i', '--identity', type=int, default=0,
                    help='the default result for no arguments '
                         '(default: 0)')
parser.add_argument('--sum', dest='accumulate', action='store_const',
                    const=sum, default=max,
                    help='Sum the integers (default: find the max).')

if __name__ == '__main__':
    args = parser.parse_args()
    print(args.accumulate(args.integers))
```

In order to document the above program:

```
.. autoprogram:: cli:parser
   :prog: cli.py
```

That's it. It will be rendered as:

1.1 cli.py

Process some integers.

```
usage: cli.py [-h] [-i IDENTITY] [--sum] N [N ...]
```

n
An integer for the accumulator.

-h, --help
show this help message and exit

-i <identity>, --identity <identity>
the default result for no arguments (default: 0)

--sum
Sum the integers (default: find the max).

If there are subcommands (subparsers), they are rendered to subsections. For example, given the following Python CLI program (say it's `subcmds.py`):

```
import argparse

parser = argparse.ArgumentParser(description='Process some integers.')
subparsers = parser.add_subparsers()

max_parser = subparsers.add_parser('max', description='Find the max.')
max_parser.set_defaults(accumulate=max)
max_parser.add_argument('integers', metavar='N', type=int, nargs='+',
                        help='An integer for the accumulator.')

sum_parser = subparsers.add_parser('sum', description='Sum the integers.')
sum_parser.set_defaults(accumulate=sum)
sum_parser.add_argument('integers', metavar='N', type=int, nargs='+',
                        help='An integer for the accumulator.')

if __name__ == '__main__':
    args = parser.parse_args()
    print(args.accumulate(args.integers))
```

```
.. autoprogram:: subcmds:parser
   :prog: subcmds.py
```

The above reStructuredText will render:

1.2 subcmds.py

Process some integers.

```
usage: subcmds.py [-h] {max,sum} ...
```

-h, --help
show this help message and exit

1.2.1 subcmds.py max

Find the max.

```
usage: subcmds.py max [-h] N [N ...]
```

n

An integer for the accumulator.

-h, --help

show this help message and exit

1.2.2 subcmds.py sum

Sum the integers.

```
usage: subcmds.py sum [-h] N [N ...]
```

n

An integer for the accumulator.

-h, --help

show this help message and exit

.. autoprogram:: module:parser

It takes an import name of the parser object. For example, if `xyz` variable in `abcd.efgh` module refers an `argparse.ArgumentParser` object:

```
.. autoprogram:: abcd.efgh:xyz
```

The import name also can evaluate other any Python expressions. For example, if `get_parser()` function in `abcd.efgh` module creates an `argparse.ArgumentParser` and returns it:

```
.. autoprogram:: abcd.efgh:get_parser()
```

It also optionally takes an option named `prog`. If it's not present `prog` option uses `ArgumentParser` object's `prog` value.

Additional Options for `.. autoprogram::`

New in version 0.1.3.

:maxdepth: `##` Only show subcommands to a depth of `##`.

:no_usage_codeblock: Don't put the usage text in a `.. codeblock:: console` directive.

:start_command: `subcommand` Render document for the given subcommand. `subcommand` can be a space separated list to render a sub-sub-...-command.

:strip_usage: Removes all but the last word in the usage string before the first option, replaces it with `'...'`, and removes an amount of whitespace to realign subsequent lines.

CHAPTER 3

Author and license

The *sphinxcontrib.autoprogram* is written by [Hong Minhee](#) and distributed under BSD license.

The source code is maintained under the [GitHub repository](#):

```
$ git clone git://github.com/sphinx-contrib/autoprogram.git
```


4.1 Version 0.1.4

Released on February 27, 2018.

- Fixed a `:rst:dir'.. autoprogram::'` bug that raises `AttributeError` during build without `:no_usage_codeblock:` option on Python 2. [Bitbucket issue #168, Bitbucket issue #169]
- Fixed an issue with Sphinx 1.7 which removed `sphinx.util.compat`. [#1, #2 by Zach Riggle]

4.2 Version 0.1.3

Released on October 7, 2016.

- Fixed a bug that descriptions with `RawTextHelpFormatter` had been incorrectly formatted. [Bitbucket PR #123 by Aaron Meurer]
- Fixed crash when metavar is a tuple (i.e. for `nargs > 1`). [Bitbucket PR #112 by Alex Honeywell]
- Fixed usage string for subcommands (subcommands were previously showing the top-level command usage). [Bitbucket PR #112 by Alex Honeywell]
- Added *new options* to `.. autoprogram:: directive:` [Bitbucket PR #112 by Alex Honeywell]
 - `maxdepth`
 - `no_usage_codeblock`
 - `start_command`
 - `strip_usage`
- Fixed suppressed arguments (using `argparse.SUPPRESS` flag) to become ignored. [Bitbucket issue #166]

4.3 Version 0.1.2

Released on August 18, 2015.

- Fixed crash with empty fields. [[Bitbucket issue #110](#)]
- Fixed `ImportError` with non-module Python scripts (i.e. files not ending with `.py`). [[Bitbucket PR #101](#) by Matteo Bachetti]

4.4 Version 0.1.1

Released on April 22, 2014.

- Omit metavars of `store_const/store_true/store_false` options.
- Sort subcommands in alphabetical order if Python 2.6 which doesn't have `collections.OrderedDict`.

4.5 Version 0.1.0

Released on March 2, 2014. The first release.

S

`sphinxcontrib.autoprogram`, 1

Symbols

`-sum`
cli.py command line option, 4

`-h, -help`
cli.py command line option, 4
subcmds.py command line option, 4
subcmds.py-max command line option, 5
subcmds.py-sum command line option, 5

`-i <identity>, -identity <identity>`
cli.py command line option, 4

A

autoprogram (directive), 5

C

cli.py command line option
-sum, 4
-h, -help, 4
-i <identity>, -identity <identity>, 4
n, 4

N

n
cli.py command line option, 4
subcmds.py-max command line option, 5
subcmds.py-sum command line option, 5

S

sphinxcontrib.autoprogram (module), 1

subcmds.py command line option
-h, -help, 4

subcmds.py-max command line option
-h, -help, 5
n, 5

subcmds.py-sum command line option
-h, -help, 5
n, 5